Computer Science Department

TECHNICAL REPORT

A Time-Independent Definition of Software Reliability

Stewart N. Weiss and Elaine J. Weyuker

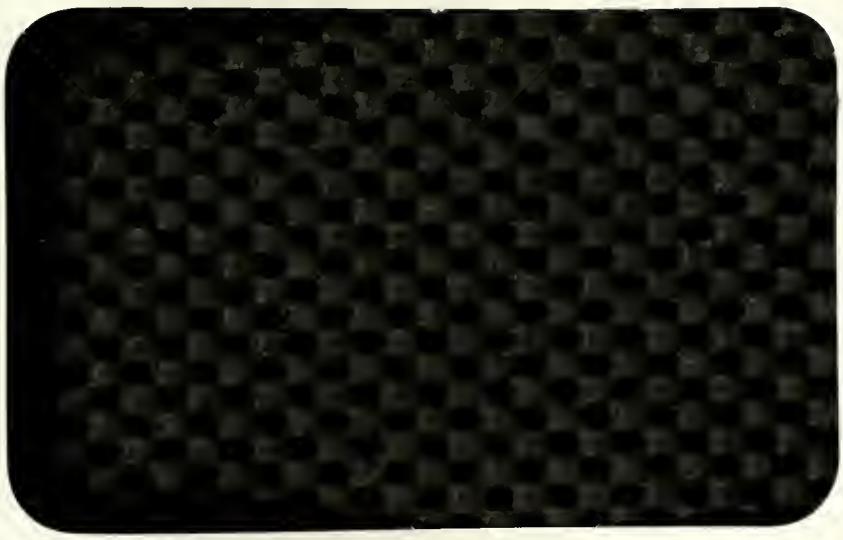
Technical Report #146

January 1985

NYU COMPSCI 146 c.2
Weiss, Stewart N
A time-independent
definition of

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



NEW YORK UNIVERSITY

A Time-Independent Definition of Software Reliability

Stewart N. Weiss and Elaine J. Weyuker

Technical Report #146

January 1985

ABSTRACT

A definition of software reliability is proposed in which reliability is treated as a generalization of the probability of correctness of the software in question. The definition is parametrized by the distribution characterizing the operational environment. It is shown that the definition can be used to provide many natural models of reliability by varying an integer parameter, and that it may be approximated reasonably using well-chosen test sets. It is proved that, under fairly weak conditions, one can not hope to measure reliability exactly by using finite test sets.

Index Terms – Software reliability, operational distribution, input domain based model.

C 2

A Time-Independent Definition of Software Reliability

Stewart N. Weiss and Elaine J. Weyuker

Introduction

Software reliability has been described as "a metric which is the probability of operational success of the software"[17]. Because the measurement of software reliability is in principle the modelling of a deterministic process by a probabilistic one, the problem arises as to how to model the uncertainty of the ultimate outcome. When the random variable is chosen to be time, we refer to the implicit definition of reliability as time-dependent. When the random variable is otherwise defined, we will say the definition is time-independent.

Most time-dependent definitions of program reliability express the idea that the reliability of a program is the probability that a software error which causes a discrepancy from specified requirements, by more than specified tolerances, in a specified environment does not lead to a failure during a specified exposure period [1,5,7,10,12,17,18]. Time-independent definitions attempt to capture the idea that the reliability of a program is the probability that an arbitrary run of the program will produce the specified results [2].

Time-dependent definitions provide a straightforward approach to measuring reliability; the indices most commonly derived in such an approach include the mean time to failure, number of remaining errors, and probability of failure in a time interval [0,t]. While these indices are relatively easy to compute in many cases, there is one major weakness in their uses as measures of reliability : a piece of software is neither more nor less *dependable* today than it was yesterday if neither it nor its intended use nor its operational environment has changed. Perhaps our confidence in it has changed, but not its inherent reliability. However, the reliability indices described above frequently change with the passing of time or the discovery of errors in the software.

Although there has been a great deal of debate as to which among the time based models is the most reasonable [6,11,13,15,16], there has been little theoretical justification of their validity. Historically, these models have been judged in part on statistical analysis, empirical results, and ease of use. Recently, Iannino et al [9] proposed a set of criteria by which to judge software reliability models, particularly the time-dependent ones, and recommended that these criteria be used to compare and assess these models. This is an important step towards providing a firm basis for the use of these models.

We consider here the time-independent aspects of reliability, and, unlike time-based models, our model treats the operational input as its random variable. We believe that the

This research was supported in part by the National Science Foundation under grant MCS-82-01167.

reliability of software is not a measure of our confidence in it, but rather a measure of its trustworthiness for the job it is intended to do. As such, we believe that any reasonable definition should incorporate :

1) measurement of likelihood, and

2) measurement of the discrepancy between actual program behavior and intended program behavior, and

3) parametrization of the external environment influencing program behavior.

We therefore construct a family of metrics to achieve (2). The metrics in this family differ in the extent to which they consider important the magnitudes of program discrepancies. We then show that by an appropriate choice of parameter we can find a metric in this family to model suitably a given notion of distance. Finally, we use the operational distribution to transform the metric spaces created into probability spaces, thereby achieving both (1) and (3). We show that the definitions model reliability in a natural way. In addition, we show that using finite test sets to estimate software reliability cannot in general yield exact reliability based on this definition.

Preliminaries

The *domain* of a program **P** is the set of all input values for which the execution of **P** terminates. Without loss of generality, input and output values are restricted to **N**, the non-negative integers, and hence both the domain and the range of a program are subsets of **N**. It is well-known [5] that this model of computation is equivalent to one in which the objects of computation are finite-length strings over a fixed finite alphabet.

It is assumed that all programs are written in some fixed, but unspecified, programming language, $\mathcal{L}$, which is powerful enough to compute any partially computable function. We write $\Phi = \{\phi_i\}$ to be an enumeration of all of the legal programs in $\mathcal{L}$.

We sometimes abuse the distinction between a program **P** and the partially computable function it computes. If program **P** halts on input n, we write $\mathbf{P}(n)\downarrow$; if not, we write $\mathbf{P}(n)\uparrow$. We write **P**(n) to mean the output of **P** on input n, if $\mathbf{P}(n)\downarrow$; otherwise, we denote the result by ∞. We extend **N** by the inclusion of the "point" ∞, and we write $\mathbf{N}^* = \mathbf{N} \cup \{\infty\}$. We endow $\mathbf{N}^*$ with the usual ordering and arithmetic properties, i.e., $(\forall n \in \mathbf{N})(n < \infty)$ and $n + \infty = \infty$, etc.

We write $\mathbf{P}(n) \stackrel{*}{=} \mathbf{Q}(n)$ if either $[\mathbf{P}(n)\downarrow \wedge \mathbf{Q}(n)\downarrow \wedge \mathbf{P}(n) = \mathbf{Q}(n)]$ or $[\mathbf{P}(n)\uparrow \wedge \mathbf{Q}(n)\uparrow]$. Two programs **P** and **Q** are equivalent, denoted $\mathbf{P} \equiv \mathbf{Q}$, if $(\forall n \in \mathbf{N})(\mathbf{P}(n) \stackrel{*}{=} \mathbf{Q}(n))$.

There remains the problem of defining a specification **S** of a program. A specification is supposed to prescribe the output of the program for each input. However, three cases can arise for a given input n :

1) no output is specified.

2) exactly one output is specified.

3) more than one output is specified.

We take the position that if case (1) occurs for a given input n, then that input is not an illegal value, but a "don't care"; such inputs are outside the sphere of interest of the specification. We collect all such n into the set $\mathbf{U} = \{ n \mid \mathbf{S} \text{ is unspecified at } n \}$. If **P** is some program and $n \in \mathbf{U}$, we define **P**(n) to be a correct value for **S**(n). Note that we do not say **P**(n) = **S**(n), since the transitivity of "=" does not hold. Illegal inputs are considered to produce some specified error message and hence some defined value. In the sequel we use **P** and **Q** to denote programs and **S** to denote a program specification.

Although it is certainly possible that a specification defines any of a class of outputs to be acceptable, we require that for a given input, either case (1) or (2) holds and, that given any input, we can effectively decide which case does hold.

With the above understandings, a specification **S** is a partial function over **N**, with computable domain $\mathbf{D} = \mathbf{N} - \mathbf{U}$. Our stronger assumption is that **S** defines a partially computable function over **N**, and we henceforth identify **S** with that function.

Suppose we have a specification **S** with domain **D**, intended for use in a given application "environment". Each element $n \in \mathbf{N}$ has an associated probability, say p(n), of being used as an input to the implementation in this environment, and this probability is not necessarily dependent on membership in **D**. In the literature, this probability distribution has been called the *operational profile* of the software [5]. Each particular environment defines some such underlying distribution of **N**, and we assume here that the distribution is independent of the implementation program (which may not always be the case.)

A specification **S** might be implemented in more than one "environment", and so there may be several distributions associated with **S**. If we have a program intended to implement specification **S**, it thus makes sense to speak of the reliability of the program with respect to both **S** and the operational distribution of the actual environment in which the program is used. It seems reasonable to us to incorporate this distribution into the measure of reliability in a definite way. We will generally use p to denote the probability function defining the operational distribution.

The fact that we may not have knowledge of a particular distribution does not imply that it does not exist. In cases when no information is available, p would normally be chosen to be a uniform distribution. But any available information should be used to tailor the measurement of reliability to the actual environment. It is this thought, together with the preceding discussion, that moves us to make the following :

Assertion : To each specification **S** and environment in which **S** is implemented, there is a unique probability function p, such that for each $n \in \mathbf{N}$, $p(n)$ is the likelihood that n is used as an input to the implementation of the specification in that environment.

We emphasize that we are not claiming here that p is computable; we consider the question of computability later. The intention here is to use the distribution as a parameter of the measure of the reliability of a program. To do so we incorporate p into our definition of a family of metrics defined on the set of equivalence classes of Φ with respect to an equivalence we define below.

Given a distribution p, we call a value $n \in \mathbf{D}$ *possible with respect to* p, or *p-possible*, if $p(n) > 0$. If no confusion can arise, we simply call n *possible*. Given two programs **P** and **Q**, we will say that **P** and **Q** are *equivalent with respect to* p, or *p-equivalent*, denoted $\mathbf{P} \underset{p}{=} \mathbf{Q}$, if $(\forall n \in \mathbf{N})\ (\ (p(n) > 0) \wedge (\mathbf{P}(n) \stackrel{*}{=} \mathbf{Q}(n))\ )$. In other words, **P** and **Q** are p-equivalent if they agree on all p-possible inputs. It follows immediately from the definitions that $\mathbf{P} = \mathbf{Q}$ if and only if for all distributions p, $\mathbf{P} \underset{p}{=} \mathbf{Q}$.

Definition: We call two programs **P** and **Q** *p-compatible*, and we write $\mathbf{P} \underset{p}{\sim} \mathbf{Q}$, if $(\forall n \in \mathbf{N})\ ((p(n) > 0) \Rightarrow (\mathbf{P}(n)\cdot \Leftrightarrow \mathbf{Q}(n)\cdot))$.

Note that p-compatibility is an equivalence relation on the set Φ, the enumeration of all legal programs in $\mathcal{L}$.

The Definition and its Properties

Suppose that **P** and **Q** are programs in Φ which halt on all inputs in **N**. A natural way to measure the discrepancy between **P** and **Q** on input n is $|\mathbf{P}(n) - \mathbf{Q}(n)|$, the usual Euclidean distance. However, if either **P** or **Q** diverges on input n then this distance function is undefined. Clearly, we would like a metric on the output space ($\mathbf{N}^*$) of programs in $\mathcal{L}$ having the property that it is always defined, even when one of its arguments is undefined (∞). Futhermore, there are times when our only concern in comparing **P**(n) and **Q**(n) is whether or not they are equal. This amounts to using a binary valued metric on $\mathbf{N}^*$. In short, it is useful to have a family of metrics that is large enough to characterize many different notions of distance. For the above reasons we make the following :

Definition: Let $\alpha \in \mathbf{N}$, $\alpha \geq 1$. For any $n \in \mathbf{N}$, and any programs **P** and **Q** we write

$$|P(n),Q(n)|_\alpha = \begin{cases} min(\alpha, |P(n)-Q(n)|) & \text{if } P(n)\cdot \wedge Q(n)\cdot \\ 0 & \text{if } P(n)\uparrow \wedge Q(n)\uparrow \\ \alpha & \textit{otherwise} \end{cases}$$

We shall call this the *α-discrepancy* between **P** and **Q** at n. When $\alpha = 1$, the definition reduces to

$$|P(n),Q(n)|_1 = \begin{cases} 0 & \text{if } P(n) \stackrel{*}{=} Q(n) \\ 1 & \textit{otherwise} \end{cases}$$

This is the usual "binary" interpretation of the correctness of one computation with respect to another. When α is very large, the definition approximates the Euclidean distance defined

on the set $\{ P(n) \mid n \in \mathbf{N} \wedge P \in \Phi \}$.

Definition: Let p define a distribution of **N**. Define $\delta_\alpha^p : \Phi \times \Phi \to [0,\alpha]$ by

$$\delta_\alpha^p(P,Q) = \sum_{n=0}^{\infty} p(n) \cdot |P(n),Q(n)|_\alpha$$

For programs **P** and **Q**, $\delta_\alpha^p(P,Q)$ is the expected value of the α-discrepancy with respect to the distribution of the input domain defined by p. If **Q** is a program specification and **P**, an intended implementation of **Q**, then $\delta_\alpha^p(P,Q)$ should be thought of as an approximation to the expected "error" in **P**. When $\alpha = 1$, this corresponds to the probability, with respect to the operational distribution, of that subset of **N** on which **P** and **Q** differ. When α is very large, it is more like an average discrepancy between **P** and **Q**, weighted by the operational profile.

Definition: Let X be any set and let x, y be elements of X. A function $d : X \times X \to \mathbf{R}$, the real numbers, is called a *metric* on the set X if

(a) $d(x,y) > 0$ if $x \neq y$; $d(x,x) = 0$

(b) $d(x,y) = d(y,x)$

(c) $d(x,y) \leq d(x,z) + d(z,y)$ for all $z \in X$.

It is straightforward to show

Theorem 1: For any $\alpha \geq 1$ and any p. δ_α^p is a metric on the set of equivalence classes of Φ with respect to p.

We next consider the following questions:

(1) How does the choice of α effect the *meaning* of the metric ?

(2) Does this family of metrics enable us to characterize different notions of proximity of partially computable functions?

As a consequence of Theorem 1, for any partially computable functions **P** and **Q** and any $\alpha_1 > 1$ and $\alpha_2 > 1$, $\delta_{\alpha_1}^p(P,Q)=0 \Leftrightarrow \delta_{\alpha_2}^p(P,Q)=0$. In other words, "identity" is invariant with respect to the choice of α. Another way to view this is that the number of distinct points in the metric space of Theorem 1 is independent of the choice of α. We can further show that :

Lemma 2: If **P** and **Q** are partially computable functions, there is a number **M** such that for all $\alpha \geq M$,

$$\delta_{\alpha}^{p}(P,Q)=\delta_{M}^{p}(P,Q)$$

if and only if

(1) $(\forall n)\ [\ [P(n)\downarrow \Leftrightarrow Q(n)\downarrow] \vee p(n) = 0\ ]$ and

(2) $(\exists K)\ (\forall n)[\ [\ P(n)\downarrow \wedge Q(n)\downarrow \Rightarrow |P(n) - Q(n)| \leq K\ ]\ \vee\ [p(n) = 0]\ ]$

The usefulness of this lemma is that it shows that whenever two programs are defined on the same domain and the difference between the values produced as a result of their halting computations on all inputs is *bounded*, then for sufficiently large α, the choice of α does not effect the distance between them as measured under the δ_{α}^{p}-metric. We note here that in any fixed programming environment, condition (2) always holds, so that whenever condition (1) holds also, such M exists, and would be a natural choice to use as an upper bound for α in this situation.

In many situations, we consider any deviation of the program's output from the specified output as an error. In such cases, $\alpha = 1$ models our intuitive notion of the discrepancy. However, for other situations, we are willing to tolerate "small" discrepancies of the output with respect to its specification, and thus we measure the reliability of the program based on the overall effect of these discrepancies. Such might be the case with approximation algorithms. In these cases, a large value of α, say $\alpha=2^{M}$, where M is the word size of the machine, would model this notion reasonably.

The intuitive definition of the reliability of a program with respect to a specification **S** in a given environment is that it is the probability that on an arbitrary input n, **P** will yield the same result as **S**. This is essentially how Brown and Lipow, for example, define software reliability in [2], and we formalize it in the following

Definition: Let **S** be a specification with domain **D** and let **P** be a program intended to compute the function **S**. Then, if p is a probability function characterizing the domain of **P**, we define the *reliability* of **P** with respect to **S**, denoted R[p,S](P):

$$R[p,S](P)=1-\delta_{1}^{p}(P,S_P) \quad \textit{where} \quad S_P(n) = \begin{cases} S(n) & \text{if } n\in D \\ P(n) & \text{if } n\in U \end{cases}$$

Intuitively, this is the probability of correctness with respect to the distribution of the input domain. We need to compare **P** to $\mathbf{S}_{\mathbf{P}}$ because we *"don't care"* about $n \in U$. An example will demonstrate :

Example 1

Let M be the word size of a given machine and let p be given by

$$p(n) = \begin{cases} 2^{-M} & \text{if } 0 \leq n < 2^{M} \\ 0 & \textit{otherwise} \end{cases}$$

i.e., the inputs are uniformly distributed over the range of representable natural numbers. Let **S** specify the following function:

$$S(n) = \begin{cases} n & \text{if } 2^{M-1} \le n < 2^M \\ 0 & \textit{otherwise} \end{cases}$$

Let **P** be written to compute

$$P(n) = \begin{cases} n-1 & \text{if } 2^{M-1} \le n < 2^M \\ 0 & \textit{otherwise} \end{cases}$$

and suppose another program **Q** has been written to compute

$$Q(n) = \begin{cases} n & \text{if } 2^{M-1} < n < 2^M \\ 0 & \textit{otherwise} \end{cases}$$

Thus, **P** differs from **S** by a "small" amount for many inputs, while **Q** differs from **S** at only one point, but by a "large" amount.

Then, by the definition,

$$R[p,S](P) = 1 - \frac{1}{2^M} \cdot \sum_{2^{M-1}}^{2^M-1} 1 = \frac{1}{2}$$

$$R[p,S](Q) = 1 - \frac{1}{2^M}$$

Using this measure of reliability, **P** is deemed highly unreliable because it produces the wrong answer at half of the equally likely possible inputs. In contrast to this, **Q** would be considered much more reliable by this measure since it is correct at all but 1 of these 2^M possible inputs.

We now use the parameter α to generalize the above definition:

Definition : With **S**, **P**, **D**, p, and $\mathbf{S_P}$ as above, we define, for each $\alpha \ge 1$, the α-*reliability* of **P** with respect to **S**, denoted R[α,p,S](P):

$$R[\alpha,p,S](P) = 1 - \frac{\delta_\alpha^p(P,S_P)}{\alpha}$$

The α-reliability of **P** with respect to **S** corresponds to the expected value of the α-discrepancy between **P** and **S** with respect to the input distribution defined by p. As α increases we weight the effect of an error by the amount of discrepancy. Thus, one is penalized "only a little" for small discrepancies (relative to α). Note that the maximum penalty for an error on input n with this generalized definition is still p(n), which occurs when $|P(n),S_P(n)|_\alpha = \alpha$. We should expect that with such relaxed standards, programs should be perceived as more "reliable" since we are considering fewer errors "important". The next lemma demonstrates that this is, in fact, the case.

Lemma 3: For given program **P**, distribution p, and specification **S** with domain **D**, if $1 \le \alpha \le \beta$ then $R[\alpha,p,S](P) \le R[\beta,p,S](P)$.

Thus, the reliability function is a non-decreasing function of the parameter α, when **P** and **S** remain fixed. As $\alpha \to \infty$, however, the reliability of an arbitrary program does not necessarily approach 1; it is not difficult to show that :

Theorem 4: If **S** is a specification with domain **D**, **P** is a program, and p is a probability function, and if $D_2 = \{ n \mid [P(n)\downarrow \wedge S_P(n)\uparrow] \vee [P(n)\uparrow \wedge S_P(n)\downarrow] \}$ then

$$\lim_{\alpha \to \infty} R[\alpha,p,S](P) = 1 - p(D_2).$$

The proof is a straightforward application of the definitions.

If **P** is a program which is p-compatible with $\mathbf{S}_p$, then, with D_2 as defined in Theorem 4, $p(D_2) = 0$. The theorem shows that, in this case, the reliability of **P** converges to 1 as $\alpha \to \infty$. This is, of course, consistent with our intuition. As $\alpha \to \infty$, the only errors which are "important" are those inputs for which the program or specification halts while the other does not. If there are no such p-possible inputs, ($p(D_2) = 0$) then the program is "highly reliable" as it does not contain any errors in the designated set. More generally, it shows that the reliabilities of all programs in a given p-compatible equivalence class converge to the same point in the interval [0,1] as $\alpha \to \infty$. This behavior makes it difficult to compare reliabilities obtained with different values of α.

We return to Example 1, to illustrate the role of α in the generalized notion of reliability.

Example 2

Let S, p, P, and Q be those of Example 1, and consider the generalized reliability functions of P and Q:

$$R[\alpha,p,S](P) = 1 - \frac{1}{2^M \cdot \alpha} \cdot \sum_{n=2^{M-1}}^{2^M-1} 1 = 1 - (2\alpha)^{-1}$$

$$R[\alpha,p,S](Q) = \begin{cases} 1 - 2^{-M} & \text{if } \alpha < 2^{M-1} \\ 1 - (2\alpha)^{-1} & \text{if } \alpha \geq 2^{M-1} \end{cases}$$

When $\alpha = 1$, R[α,p,S](P) simply measures the frequency of error, relative to the input distribution. As α grows, the magnitudes of the errors play an increasingly important role. Notice that in this example, the total discrepancies of both **P** and **Q** are the same, 2^{M-1}. When α is less than this number, the reliability of **P** is measured to be less than that of **Q**. When α is greater than or equal to this number, the two programs have the same measure of reliability. □

Example 3

The required program must compute the function

$$S(n) = trunc\left(100000 \cdot \exp\left(\frac{n}{1000} \right) \right)$$

for all n in the range [0, 9999], where $\exp(x)$ is defined, for all rational x, by its infinite expansion, $\sum_{k=0}^{\infty} \frac{x^k}{k!}$. Since the specification requires no more than five decimal places of precision in the decimal expansion of $\exp(n/1000)$, S(n) is both well-defined and computable. Program P performs the calculation using the algorithm

$$P(n) = trunc\left(100000 \cdot 2.7183^{\frac{n}{1000}}\right)$$

Program Q performs the calculation using the algorithm

$$Q(n) = trunc\left(100000 \cdot \sum_{k=0}^{8} \frac{n^k}{1000^k \cdot k!}\right)$$

Our objective is to assign a measure of accuracy to each of the two methods of performing the calculation under the uniform distribution on the integer interval [0,9999]. To this end, we select a set of 50 test points, T = {0, 200, 400, ...,9800}, and we assume that the computations with these points represent fairly the true reliability. The following table shows the reliabilities of the programs P and Q restricted to the test set, using the definition with different values for α, and with the uniform distribution on the set $\{200n \mid 0 \le n \le 49, n \in N\}$.

α	R[α,p,S](P)	R[α,p,S](Q)
1	.020	.100
10	.020	.144
100	.063	.181
1000	.171	.227
10000	.328	.289
100000	.511	.366
1000000	.707	.462
10000000	.912	.582
100000000	.991	.727

As one would expect, the table reflects the fact that the finite Taylor polynomial is an asymptotically poorer approximation than the iterated power method. (Here, both programs were written in C and run under UNIX version 4.2 BSD on a VAX-11/750.) The table shows that Q generates fewer discrepancies than P, since when $\alpha = 1$, Q's reliability is greater. It shows that these discrepancies must be of greater total magnitude though, since when $\alpha \ge 10000$, Q is less reliable by our definition. In addition, it shows that in both cases there are many discrepancies, since when $\alpha = 1$, the reliabilities of both P and Q are low. □

Our final example is of a situation in which the input domain is governed by a form of Pareto distribution.

Example 4

Let **S** be a specification of a program to compute income tax, to the nearest dollar, when given as input an adjusted gross yearly income, also to the nearest dollar. Suppose that there are two programs, **P** and **Q**, written to satisfy **S**, and that it has been determined that neither is correct. In fact, we assume that it has been determined that the programs are incorrect according to the following patterns :

$$P(n) = \begin{cases} S(n) + 50 & \text{if } n \in \{\, 10000 + 10k \mid 1 \le k \le 2000 \,\} \\ S(n) + 100 & \text{if } n \in \{\, 30000 + 100k \mid 1 \le k \le 99700 \,\} \\ S(n) & \textit{otherwise} \end{cases}$$

$$Q(n) = \begin{cases} S(n) + 1 & \text{if } 1 \le n \le 10000 \\ S(n) + 5 & \text{if } 10000 < n \le 20000 \\ S(n) + 10 & \text{if } 20000 < n \le 30000 \\ S(n) & \textit{otherwise} \end{cases}$$

The operational distribution is given by

$$p(n) = \begin{cases} 0 & \text{if } n = 0 \\ \zeta(2)^{-1} \cdot 10^{-4} \cdot \left\lceil \dfrac{n}{10^4} \right\rceil^{-2} & \text{if } n > 0 \end{cases}$$

where $\zeta(s)$ is the Riemann zeta function of s, for s > 1, which is defined by :

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}$$

We compute the α-reliabilities of **P** and **Q** :

$$R[\alpha,p,S](P) = 1 - \frac{1}{\alpha} \cdot \left(\sum_{n=10001}^{20000} p(n) \cdot \min(\alpha,50) + \sum_{n=20001}^{30000} p(n) \cdot \min(\alpha,100) \right)$$

$$= 1 - \frac{1}{\alpha} \cdot \left(\zeta(2)^{-1} \cdot 10^{-4} \right) \cdot \left(\sum_{k=1}^{20} \frac{1000}{k^2} \cdot \min(\alpha,50) + \sum_{k=21}^{1000} \frac{100}{k^2} \cdot \min(\alpha,100) \right)$$

$$\approx 1 - \frac{1}{\alpha} \cdot \left(\frac{6}{10^4 \cdot \pi^2} \right) \cdot \left(1596.2 \cdot \min(\alpha,50) + 28.38 \cdot \min(\alpha,100) \right)$$

$$\approx 1 - \frac{.6079}{\alpha \cdot 10^4} \cdot \left(1596.2 \cdot \min(\alpha,50) + 28.38 \cdot \min(\alpha,100) \right)$$

and

$$R[\alpha,p,S](Q) = 1 - \frac{1}{\alpha} \cdot \left(\sum_{n=1}^{10000} p(n) \cdot \min(\alpha,1) + \sum_{n=10001}^{20000} p(n) \cdot \min(\alpha,5) + \sum_{n=20001}^{30000} p(n) \cdot \min(\alpha,10) \right.$$

$$= 1 - \frac{1}{\alpha} \cdot \left(\zeta(2)^{-1} \cdot 10^{-4} \right) \cdot \left(\sum_{k=1}^{10000} \min(\alpha,1) + \sum_{k=10001}^{20000} \frac{1}{4} \cdot \min(\alpha,5) + \sum_{k=200001}^{30000} \frac{1}{9} \cdot \min(\alpha,1 \right.$$

$$= 1 - \frac{1}{\alpha} \cdot \left(\frac{6}{10^4 \cdot \pi^2}\right) \cdot \left(10^4 + \frac{10^4}{4} \cdot \min(\alpha,5) + \frac{10^4}{9} \cdot \min(\alpha,10)\right)$$

$$\approx 1 - \frac{.6079}{\alpha} \cdot \left(1.0000 + 0.2500 \cdot \min(\alpha,5) + .1111 \cdot \min(\alpha,10)\right)$$

The above functions may be defined piecewise as follows :

$$R[\alpha,p,S](P) \approx \begin{cases} .9012 & \text{if } \alpha \leq 50 \\ .9983 - 4.852 \cdot \alpha^{-1} & \text{if } 50 \leq \alpha \leq 100 \\ 1.000 - 5.024 \cdot \alpha^{-1} & \text{if } 100 \leq \alpha \end{cases}$$

$$R[\alpha,p,S](Q) \approx \begin{cases} .7805 - .6079 \cdot \alpha^{-1} & \text{if } \alpha \leq 5 \\ .325 - 1.368 \cdot \alpha^{-1} & \text{if } 5 \leq \alpha \leq 10 \\ 1.000 - 2.043 \cdot \alpha^{-1} & \text{if } 10 \leq \alpha \end{cases}$$

By solving the inequalities we see that when $\alpha \leq 20$, the reliability of **P** is greater than that of **Q**, while when $\alpha \geq 21$, **Q** is deemed more reliable than **P**. In particular, we show some results for selected values of α :

α	R[α,p,S](P)	R[α,p,S](Q)
1	.901	.173
10	.901	.796
20	.901	.898
50	.901	.959
100	.950	.980

□

The preceding examples demonstrate that the relative reliabilities of **P** and **Q** are governed by the choice of α. Hence, the choice of α should be guided by the characterization of reliability desired. That is, one would decide upon the extent to which reliability should measure the "sizes" of errors and the frequency of errors. Therefore, one is certainly able to characterize many notions of reliability within this family of reliability measures. In fact, one can show that :

Lemma 5: Given any specification **S**, probability function p, and $\alpha \geq 1$, there are programs **Q** and **Q'** such that

$$R[\alpha,p,S](Q) > R[\alpha,p,S](Q') \textit{ but } R[\alpha',p,S](Q') \leq R[\alpha',p,S](Q') \textit{ whenever } \alpha' > \alpha \quad \square$$

Measuring Reliability

It is true that, since δ_α^p is not a computable functional, R[α,p,S] is not a computable functional. However, we do not let this fact prevent us from *using* the definition to model reliability. In general we hope to *measure* reliability, and applying the definition clearly falls

short in this respect. However, this is not an uncommon situation in mathematics. For example, although the definition of the definite integral falls short in much the same way, we don't let that stop us from calculating integrals.

In order to discuss using the definition we must clarify some technical points about computable functions. Recall that the range of any partially computable function in our model is a subset of the natural numbers. Whenever β is a rational number, what we mean by the statement that program **P** computes $\mathbf{P}(n) = \beta$ is that if $\beta = q/r$ where $\gcd(q,r) = 1$, then the output of program P on input n is the number $\langle q,r \rangle$ for some pairing function $\langle \, , \rangle$. When we say that a real-valued function f is computable, we mean that whenever $f(n)$ is irrational we can use an algorithm for f to compute an arbitrarily close rational approximation to $f(n)$. More formally, we say that there is a total function, o: $\mathbf{N} \rightarrow \mathbf{R}$, such that $o(m) \rightarrow 0$ as $m \rightarrow \infty$, and a rational computable function $\hat{f}$: $\mathbf{N} \times \mathbf{N} \rightarrow \mathbf{Q}$, such that $(\forall n,m \in \mathbf{N}) \ (|\hat{f}(n,m) - f(n)| < o(m))$. It is in the above sense that we will mean that a function f is a computable real function.

Suppose that we must determine the reliability of a program **P** with respect to its specification S in a given environment. We choose some finite set T of values from the domain **D** of S and we use these values as program inputs. Assuming that we can determine the values S(t) and P(t) [†] for each t selected, we can measure, for any α, the α-discrepancies that arise. We must then assign, for each $t \in T$, a "weight", $p_T(t)$, corresponding to the probability that, in any sample space of size $|T|$, t occurs as an input to **P**, in such a way that the reliability of **P** when restricted to T will equal its reliability over the full domain. How can we arrive at such an assignment?

If we choose the set T independently of any knowledge of the operational distribution of the inputs, then we must impose a distribution, given by p_T, on the set T. If we have some knowledge of the actual distribution then we may be able to approximate it on T in such a way that T can "stand in" for the full domain. For instance, suppose that $\gamma = \sum_{t \in T} \tilde{p}(t)$, where $\tilde{p}(t)$ is some approximation to p(t). Then we can set $p_T(t) = \tilde{p}(t)/\gamma$ for $t \in T$, and $p_T(t) = 0$ otherwise. If we have no knowledge of p, then we can simply set $p_T(t) = 1/|T|$.

If we specifically choose the set T to represent the operational distribution of the full input domain,[‡] then the weights, $p_T(t)$, are *implicitly* incorporated into the choice of T, and a different action is now required. It is then sufficient to treat T as a uniformly distributed set. This is done implicitly, for example, in the Nelson model [18], where random test cases are selected from each of the cells of a partition of the input domain. Thayer et al [18] show that this produces an unbiased estimator of the actual reliability, where the definition of reliability they use is a variant of a special case of ours (with $\alpha = 1$ and under a finite domain). Brown and Lipow provide an excellent discussion of some techniques for selecting test data to

† We can not, in general, determine algorithmically whether or not P(t)↓ so that we cannot always determine P(t); however, in many cases it is possible for us to conclude this or make an educated guess about it.

‡ For example, by choosing the inputs at random according to the probability function p.

approximate the operational distribution, and provide a quantitative measure of the degree of approximation [2].

In any case, the function p_T should be computable if we are to use it in actual calculations. Furthermore, it should be a rational function, since otherwise we would have to approximate its irrational values rationally. Given these assumptions about the probability function p_T, the question arises as to whether we can actually measure the exact reliability of an arbitrary program with respect to its specification and its operational distribution. Stated more formally, given a specification **S**, distribution p of its input domain, and program **P**, does there exist a finite set T and a rational, computable distribution p_T satisfying $p_T(t) = 0$ if $x \notin T$, such that for any $\alpha \in N$ $R[\alpha,p,S](P) = R[\alpha,p_T,S](P)$?

A result of mostly theoretical interest is that, for those specifications which do not constrain their domains to be finite and which have infinitely many p-possible inputs, the answer is negative. We show that if we have a specification **S** such that there are infinitely many elements n of its domain for which $p(n) > 0$, and that if p *is computable*, then we cannot hope to compute the exact reliability of an arbitrary program using a finite set. We prove :

Theorem 6: Let **S** be a specification with domain **D**, let p be a computable probability function such that $\mathbf{D}' = \{n \in \mathbf{D} \mid p(n) > 0\}$ is infinite, and let $\alpha \geq 1$. Then there is a program **Q** such that for any finite set T and any rational probability function p_T such that $p_T(n) = 0$ whenever $n \notin T$,

$$R[\alpha,p,S](Q) \neq R[\alpha,p_T,S](Q)$$

proof

The proof uses some elementary results from number theory. The idea is to construct effectively a subset of elements of **D'** such that the probability of this subset is an irrational number. This will be possible as a result of a sequence of lemmas we prove in the appendix. We then use this construction to produce a program **Q** whose reliability is irrational.

The following is the needed lemma :

Lemma 11 : Let $\mathbf{D} \subseteq \mathbf{N}$ be an infinite computable set. For any computable probability function p for which $p(n) > 0$ infinitely often for $n \in \mathbf{D}$, there exists a computable characteristic function χ such that $\sum_{n \in N} p(n) \cdot \chi(n)$ is irrational.

This follows from Lemma 10 of the appendix.

Recall that we assume **D** to be a computable set. Since p and **D** satisfy the conditions of Lemma 11, there exists a computable characteristic function χ such that $\sum_{n \in N} p(n) \cdot \chi(n)$ is irrational. We define a program **Q** as follows:

$$Q(n) = \begin{cases} S(n) + \alpha \cdot \chi(n) & \text{if } S(n)\downarrow \\ \infty & \text{if } S(n)\uparrow \wedge \chi(n) = 0 \\ 0 & \text{if } [S(n)\uparrow \wedge \chi(n) = 1] \vee n \in U \end{cases}$$

Clearly **Q** exists, since the function it computes is partially computable. Recalling that $S_Q(n) = S(n)$ if n $\in$ **D** and **Q**(n) otherwise, it follows that

$$(\forall n \in N) \mid Q(n), S_Q(n) \mid_\alpha = \alpha \cdot \chi(n)$$

Hence,

$$R[\alpha,p,S](Q) = \sum_{n \in N} p(n) \cdot \mid Q(n), S_Q(n) \mid_\alpha$$

$$= \alpha \cdot \left(\sum_{n \in N} p(n) \cdot \chi(n) \right)$$

is irrational. Since no finite sum of rational numbers can equal an irrational number, for any finite set of numbers, T, and any probability function p_T such that p_T is rational and $p_T(n) =$ 0 for t $\notin$ T, we must have

$$R[\alpha,p,S](Q) \neq R[\alpha,p_T,S](Q)$$

This completes the proof of the theorem. □

There is a well known result in the theory of software testing [8] that for any specification **S** and program **P** there exists a finite test set T such that *if we can effectively decide for each t $\in$ T whether or not P halts on t, then P is correct on T if and only if P is correct on all of D, where D is the domain of S*. The preceding theorem shows that this statement cannot be generalized in terms of reliability. Correctness is the special case when R[α,p,S](P) = 1 for any choice of α and any probability function p. Thus one might have hoped to say that for any specification **S**, any program **P**, and any distribution p of the input domain, there exist a choice of α and a finite set T such that

$$R[\alpha,p,S](P) = R[\alpha,p_T,S](P)$$

where p_T defines any (rational) distribution that is zero for n $\notin$ T. This theorem shows that this statement is false.

Summary

We have presented a mathematical definition of program reliability that applies to programs whose specifications are partially computable. It is not a computable definition, in that it assumes the existence of an algorithm to determine of each input, whether or not the computation on that input ever halts; nonetheless, it serves as a model to approximate.

The definition is parametrized by two parameters: an integer which can be used to impart a notion of accuracy of computation, and a distribution characterizing the input domain. Rarely is the latter known precisely.

We believe that the definition provides a theoretically sound measure of reliability, and claim that several previously published time-independent definitions of reliability are special cases of this definition. Furthermore, we show that the definition can be approximated if enough information is available, and that an important aspect in the measurement of software

reliability is the determination of the operational distribution of the subject software. We have also shown that infinite domains pose particular problems in that there does not necessarily exist a method to measure the exact reliability of a program for which there are infinitely many *possible* inputs specified.

One possible generalization of this definition of reliability would be to include a measure of the cost of an error (although this is reflected to some extent by the use of α which allows the discrimination between "big" and "small" errors). Also, since the lack of data and studies about operational profiles is frequently a stumbling block in the study of reliability, we suggest that such data be collected in an organized way, and that we begin to characterize some of the problems we encounter.

Acknowledgment

The authors thank Phyllis G. Frankl for her helpful comments. They also thank Professor Martin Davis for his helpful suggestions about the treatment of computable real-valued functions, and Professor Thomas C. Wesselkamper for his critical review and recommendations.

References

1. B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality", *Proceedings of the Second International Conference on Software Engineering*. San Francisco CA, October 1976, pp. 592-605.

2. J. R. Brown and M. Lipow, "Testing for Software Reliability", *Proceedings of the International Conference on Reliable Software*. Los Angeles CA, April 1975, pp. 518-527.

3. Martin Davis, *Computability and Unsolvability*. Dover Publications,Inc., New York, 1982.

4. M. D. Davis and E. J. Weyuker, *Computability, Complexity, and Languages*. Academic Press, New York, 1983.

5. Amrit L. Goel, "A Guidebook for Software Reliability Assessment", Technical Report No. 83-11, Syracuse University, Syracuse NY, April 1983.

6. _____, "A summary of the discussion on 'An analysis of competing software reliability models,'" *IEEE Transactions on Software Eng.*, Vol. SE-6, No. 5, Sept. 1980, pp. 501-502.

7. Herbert Hecht, "Mini-Tutorial on Software Reliability", *Proc. COMPSAC80*, IEEE, Chicago, 1980, pp. 383-385.

8. William E. Howden, "Reliability of the Path Analysis Testing Strategy", *IEEE Transactions on Software Eng.*, Vol. SE-2, No. 3, Sept. 1976, pp. 208-215.

9. A. Iannino, J. D. Musa, K. Okumoto, and B. Littlewood, "Criteria for Software Reliability Model Comparisons", *IEEE Transactions on Software Eng.*, Vol. SE-10, No. 6, Nov. 1984, pp. 687-691.

10. *IEEE Standard Dictionary of Electrical and Electronic Terms*, second edition, IEEE, New York, 1977.

11. B. Littlewood, *Comments on Moranda's Critique*, 1979, 22 pp.

12. W. H. MacWilliams, "Reliability of Large Real-Time Control Software Systems", *Record 1973 IEEE Symposium on Computer Software Reliability*, New York, April 1973, pp. 1-6.

13. P. B. Moranda, "Comments on : 'An analysis of competing software reliability models ' : G. J. Schick and R. W. Wolverton," 1979, 42 pp.

14. C. V. Ramamoorthy and Farokh B. Bastani, "Software Reliability - Status and Perspectives", *IEEE Transactions on Software Eng.*, July 1982, pp. 354-371.

15. G. J. Schick and R. W. Wolverton, "An analysis of competing software reliability models," *IEEE Transactions on Software Eng.*, Vol. SE-4, No. 2, Mar. 1978, pp.104-120.

16. _____, "An examination of some of the questions raised regarding the paper 'An analysis of competing software reliability models,' a rebuttal," 1979, 19 pp.

17. Martin L. Shooman, *Software Engineering*. McGraw-Hill Book Company, New York, 1983.

18. T. A. Thayer, M. Lipow, and E. C. Nelson, *Software Reliability*. TRW Series of Software Technology 2, North-Holland Publishing Co., New York, 1978.

19. Harry N. Wright, *First Course in Theory of Numbers*. John Wiley & Sons, Inc., New York, 1958.

Appendix

We include here the complete proof of Theorem 12:

Definition : A *Simple Continued Fraction* (*SCF*) is an expression of the form

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cdots}}} \tag{5}$$

in which $a_1 \in \mathbf{Z}$, the set of integers, and $a_i \in \mathbf{Z}^+$ for $i > 1$, and which does not necessarily terminate. If the number of terms in a *SCF* is finite, it is called a *finite SCF*, otherwise it is an *infinite SCF*.

Notation : We denote a finite *SCF* of the form (5) by $(a_1,a_2, \cdots ,a_n)$ or by

$$\frac{1}{a_1+} \; \frac{1}{a_2+} \; \cdots \; \frac{1}{a_n}$$

and an infinite *SCF* by $(a_1,a_2, \cdots)$ or by

$$\frac{1}{a_1+} \; \frac{1}{a_2+} \; \cdots$$

Convention : Suppose $x = (a_1,a_2, \cdots ,a_n)$. It is not hard to see that $x = (a_1,a_2, \cdots ,a_n-1, 1)$ also. However, by requiring that $a_n > 1$ in any finite *SCF* with n terms, we establish a canonical form, and we adhere to that form here.

Lemma 6 : To each rational number x there is a unique finite *SCF* $(a_1,a_2, \cdots ,a_n)$ in canonical form such that $x = (a_1,a_2, \cdots ,a_n)$. Conversely, each such *SCF* represents a unique rational number.

Lemma 7 : A number x is irrational if and only if there is an infinite *SCF* $(a_1,a_2, \cdots)$ such that $x = (a_1,a_2, \cdots)$. (We omit the necessary proofs of convergence here.) In this case also, the correspondence is unique.

The next lemma is easily proved by induction on the value of k.

Lemma 8 : Let $p = (a_1,a_2, \cdots ,a_{k-1}, s_k)$ and $q = (a_1,a_2, \cdots ,a_{k-1}, t_k)$, where s_k and t_k are rational numbers, and $a_i \in \mathbf{Z}^+$ for $i = 2,3,\ldots,k-1$. Then

$$\left[p < q\right] \Leftrightarrow \left[(t_k - s_k)\cdot(-1)^k < 0\right]$$

Lemma 9 : Let $x = (a_1,a_2, \cdots ,a_n)$ be any rational number. Then there is a number $\xi > 0$ such that whenever $0 < q < \xi$ then $x + q$ has no fewer than $n + 1$ terms in its canonical *SCF*. Moreover, ξ can be obtained recursively from x.

proof

Let $\xi = (a_1,a_2,\cdots,a_n + (-1)^{n+1}) - x$. Then since $(a_n + (-1)^{n+1}) - a_n)\cdot(-1)^n = -1$ it follows by Lemma 8 that $x < x + \xi$.

Let $y = (a_1,a_2,\ldots,a_{k-1},b_k,b_{k+1},b_{k+2},\ldots,b_n)$ where $1 \le k \le n$ and $a_k \neq b_k$ and otherwise the b_l are arbitrary positive integers, except that $b_n > 1$.

For $1 \le k < n$, let

$$\alpha_k = \frac{1}{a_{k+1} +} + \frac{1}{a_{k+2} +} \quad \cdots \quad \frac{1}{a_n}$$

$$\alpha'_k = \frac{1}{a_{k+1} +} + \frac{1}{a_{k+2} +} \quad \cdots \quad \frac{1}{a_n + (-1)^{n+1}}$$

$$\beta_k = \frac{1}{b_{k+1} +} + \frac{1}{b_{k+2} +} \quad \cdots \quad \frac{1}{b_n}$$

and let $\alpha_n = \beta_n = 0$ and $\alpha'_n = (-1)^{n+1}$. Note that for $k < n$, $0 < \alpha_k,\alpha'_k,\beta_k, < 1$ and

$$x = (a_1,a_2,\ldots,a_k + \alpha_k)$$

$$x + \xi = (a_1,a_2,\ldots,a_k + \alpha'_k)$$

$$y = (a_1,a_2,\ldots b_k + \beta_k)$$

for $1 \le k \le n$.

Applying a corollary of Lemma 8 repeatedly:

Case 1 : k even, k < n

$$b_k < a_k \Rightarrow b_k + \beta_k < a_k < a_k + \alpha'_k$$

$$\Rightarrow (a_1,a_2,\cdots,a_k + \alpha'_k) < (a_1,a_2,\cdots,b_k + \beta_k)$$

$$\Rightarrow x + \xi < y.$$

$$a_k < b_k \Rightarrow (a_k + \alpha_k) < b_k < (b_k + \beta_k)$$

$$\Rightarrow (a_1,a_2,\cdots,b_k + \beta_k) < (a_1,a_2,\cdots,a_k + \alpha_k)$$

$$\Rightarrow y < x.$$

Hence, in case k is even and k < n, $a_k \neq b_k$ implies $y < x$ or $x + \xi < y$.

Case 2 : k odd, k < n

$$b_k < a_k \Rightarrow (b_k + \beta_k) < a_k < (a_k + \alpha_k)$$

$$\Rightarrow (a_1,a_2,\cdots,b_k + \beta_k) < (a_1,a_2,\cdots,a_k + \alpha_k)$$

$$\Rightarrow y < x.$$

$$a_k < b_k \Rightarrow (a_k + \alpha'_k) < b_k < (b_k + \beta_k)$$

$$\Rightarrow (a_1,a_2,\cdots,a_k + \alpha'_k) < (a_1,a_2,\cdots,b_k + \beta_k)$$

$\Rightarrow$

Hence, in case k is odd and $k < n$, $a_k \neq b_k$ implies $y < x$ or $x + \xi < y$.
Thus, if $x \leq y \leq x + \xi$ then if $y = (b_1, b_2, \cdots, b_n)$, we must have $b_i = a_i$ for i = 1,2, ...,n-1.

Consider the case that $y = (a_1, a_2, \cdots, a_{n-1}, b_n)$. By Lemma 8,

$$y < x + \xi \Leftrightarrow (a_n + (-1)^{n+1} - b_n) \cdot (-1)^n < 0$$

$$\Leftrightarrow (a_n - b_n) \cdot (-1)^n < 1$$

$$\Leftrightarrow (a_n - b_n) \cdot (-1)^n \leq 0$$

$$\Leftrightarrow y \leq x.$$

This shows that regardless of the value of b, either $y \leq x$ or $x + \xi \leq y$. It should be clear that if $y = (a_1, a_2, \cdots, a_k)$ and $k < n$ then if k is even, $x + \xi < y$, and if k is odd, $y < x$. Together, the preceding results show that there is no *SCF* with fewer than n + 1 terms which lies in the interval $(x, x + \xi)$. This proves the lemma. □

Lemma 10 : Let $D = \{d_j\}_{j=0}^{\infty}$ be an infinite, computable subset of **N**. Let $\Pi = \{p_{d_j}\}_{j=0}^{\infty}$ be an infinite sequence of positive rational numbers such that $\lim_{j \to \infty} p_{d_j} = 0$. Then there is an infinite subset, $S = \{n_j\}_{j=0}^{\infty}$ in **D**, such that $\sum_{j=0}^{\infty} p_{n_j}$ is irrational. Moreover, if p_n is a computable function of n, then the characteristic function Γ_S of S is computable and

$$\sum_{j=0}^{\infty} p_{n_j} = \sum_{n=0}^{\infty} p_n \cdot \Gamma_S(n)$$

proof

For any rational number x let $\xi(x)$ be the number obtained by lemma 10. Let $|x|_f$ be the number of terms in the *SCF* representing x. If x is irrational then $|x|_f = \infty$. We define a function σ on **N** by

$$\sigma(0) = \min_d [\, d \in D \,]$$

$$\sigma(i+1) = \min_d \left[p_d < \xi\left(\sum_{j=0}^{i} p_{\sigma(j)} \right) \wedge [\, d > \sigma(i) \,] \wedge [\, d \in D \,] \right]$$

Since the sequence $\{ p_{d_j} \}$ is infinite, positive, and tends to zero , for each number d we are guaranteed to find a $d \in D$ such that p_d is as small as required. Thus the minimalization is total, and since σ is otherwise obtained from computable functions, σ is (totally) computable.

Clearly, σ is a strictly increasing function, so that if $S = \{ \sigma(i) \mid i \in \mathbf{N} \}$, then S is infinite, $S \subseteq D$, and by a standard result of computability theory, S is a computable set. Set $n_j = \sigma(j)$ for all j. Let Γ_S be the characteristic function of S, i.e.,

$$\Gamma_S(j) = [\, j \in S \,]$$

Then Γ_S is a computable predicate, and

$$\sum_{j=0}^{\infty} p_{n_j} = \sum_{n=0}^{\infty} p_n \cdot \Gamma_S(n)$$

Let

$$\tau_n = \sum_{j=0}^{n} p_j \cdot \Gamma_S(j) \quad \text{and} \quad \tau = \lim_{n\to\infty} \tau_n$$

We claim that τ is irrational. Consider the *SCF* representing τ. By our construction of the function σ, and by Lemma 9,

$$n \leq \left| \sum_{j=0}^{n} p_{n_j} \right|_f < \left| \sum_{j=0}^{n+1} p_{n_j} \right|_f$$

so clearly, as $n \to \infty$, the length of the *SCF* of τ_n increases without bound, hence $|\tau|_f = |\lim_{n\to\infty} \tau_n|_f = \infty$, which proves that τ is irrational. □

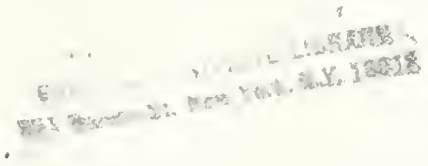
This book may be kept

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

GAYLORD 142

PRINTED IN U.S.A.



NYU COMPSCI 146 c.2
Weiss, Stewart N

A time-independent
definition of

NYU COMPSCI 146 c.2
A Weiss, Stewart N

T A time-independent
definition of

DATE DUE	BORROWER'S NAME

N.Y.U. Courant Institute of
Mathematical Sciences
251 Mercer St.
New York, N. Y. 10012